

Um Sistema de Banco de Dados Replicado Gerenciado através de *Applets* Distribuídas

André Luis Castro de Freitas¹

¹Departamento de Matemática – Fundação Universidade Federal do Rio Grande (FURG)

Caixa Postal 474 – 96201.900 – Rio Grande – RS – Brazil

dmtalcf@super.furg.br

Abstract. *A distributed and replied database it is characterized as a collection of several similar databases, logically connected through a computers net. This work has as proposal the definition, administration and implementation of a distributed and replied database that uses several databases in different servers. Therefore, the distributed transactions, width concurrency control, are executed through the same software in the different nodes, characterizing a distributed database system. The software proposes the use of interfaces through navigation web browse, applets in the programming language Java, where the structures customers and the several servers that manipulate the databases built using the RMI.*

Resumo. *Uma base de dados distribuída e replicada caracteriza-se como uma coleção de diversas bases de dados, semelhantes, interligadas logicamente através de uma rede de computadores. Este trabalho tem como proposta a definição, gerenciamento e implementação de um banco de dados distribuído replicado que utilize várias bases de dados relacionais em diferentes servidores. Portanto, as transações distribuídas, com o controle da concorrência, são executadas através do mesmo software nos diferentes nós. O software propõe a utilização de interfaces via navegador web, applets na linguagem de programação Java e as estruturas clientes e servidores que manipulam as bases de dados construídas utilizando o serviço RMI.*

1. Introdução

Os sistemas de banco de dados têm sido criados de forma independente, sem a preocupação quanto a possíveis integrações. Assim, cada base de dados pertencente a um sistema é administrada sob regras locais e autônomas. A evolução tecnológica das redes de computadores e de banco de dados propõe a modificação na maneira como os dados são processados [Lima, 1997].

A existência de diferentes SGBDs apresenta-se como natural e inevitável no contexto das grandes organizações, e cria a necessidade de um outro nível de sistema de gerência de banco de dados, capaz de prover o acesso a dados já existentes e distribuídos em redes de computadores [Chung, 1990]. Este acesso deve ser feito sem exigir mudanças locais nos banco de dados não requerendo reprogramação dos SGBDs envolvidos [RAM, 1991].

Sistemas que atendem a esses requisitos não possuem uma denominação única, e são

chamados de sistemas de banco de dados federados, de sistemas *multidatabase* ou, ainda, de sistemas de banco de dados distribuídos heterogêneos [Breitbart, 1992].

Um sistema de banco de dados distribuído heterogêneo representa a combinação de diferentes tecnologias de bases de dados com um sistema em questão, ou seja, consiste de uma coleção de nós cada qual podendo participar na execução de transações que fazem acesso a dados em um ou diversos nós de maneira transparente [Özsu, 1999].

Atualmente existem alguns bancos de dados distribuídos comerciais que apresentam características de processamento distribuído. Entre eles pode-se citar o *Oracle 9i*, o qual apresenta fragmentação horizontal primária de tabelas e índices. Essa versão demonstra total transparência na manipulação dos dados. Um segundo exemplo é o *SQL Server 2000* cuja redução de fragmentos horizontais ainda é limitada. Por fim o *IBM Informix Dynamic Server* que utiliza também fragmentação primária através de um servidor cuja responsabilidade é estender a distribuição. Mas todos esses bancos de dados são softwares proprietários e apresentam grande complexidade de desenvolvimento e alto custo de manutenção. O grande problema destes sistemas é a não conexão a outras tecnologias de bancos de dados existentes, não permitindo a heterogeneidade.

O objetivo deste trabalho é, portanto, gerenciar e coordenar as transações de um sistema de banco de dados distribuído e replicado que manipule diferentes bancos em cada um dos nós envolvidos.

Com o objetivo de atenuar a complexidade de desenvolvimento do sistema proposto optou-se pela utilização de réplicas, considerando os nós disponíveis, não levando em consideração a possibilidade de fragmentação. Acredita-se que a utilização de réplicas, apesar de resultar na duplicação de informações, contribua positivamente com o paralelismo, quando a exigência maior do sistema for para leituras e com a disponibilidade das informações, pois o sistema pode continuar o processamento apesar da identificação de falhas em alguns nós.

Conforme a figura 1, cada nó contém uma cópia do banco de dados (de diferentes fornecedores) que permanece conectado ao serviço de gerenciamento. O cliente acessa o serviço e estabelece a conexão com um dos nós, em caso de consultas, ou com todos os nós, em caso de atualizações.

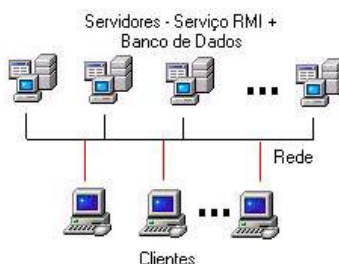


Fig. 1. Sistema de Banco de Dados Distribuído Replicado / RMI-Java

Considerando a afirmação anterior, verifica-se a necessidade de gerenciar a execução de transações que fazem acesso aos dados armazenados em um nó local. É importante ressaltar que cada transação pode ser tanto uma transação local como parte de uma transação global. Partindo deste pressuposto criou-se um serviço, disponível em cada nó, implementado a partir da linguagem Java, utilizando o mecanismo RMI – *Remote Method Invocation*.

RMI disponibiliza uma família de objetos colaborativos localizados em qualquer lugar permitindo a comunicação através de protocolos padrão através da rede. O método permite a comunicação entre máquinas virtuais Java executadas em computadores distintos [Horstmann, 2001].

Este trabalho está organizado nas seções 2 e 3. A seção 2 define e caracteriza o modelo e o ambiente da estrutura proposta. Na seção 3 é demonstrada a arquitetura do sistema, onde são verificados os lados do servidor e do cliente identificando as estruturas de conexão e controle. Após procede-se a conclusão e trabalhos futuros.

2. Modelo e Ambiente

A proposta para o modelo é a utilização de servidores *Web* distribuídos. Esta abordagem espalha a carga de pedidos entre os vários computadores conectados. A conexão do usuário ao sistema é feita pela execução de uma *applet* Java cuja cópia está disponível em cada um dos servidores. Portanto o cliente pode optar em qual nodo fará a execução, pois cada um dos componentes possui uma réplica do conteúdo a ser oferecido por esse servidor *Web*.

A *applet* Java comunica-se com um serviço disponível localmente o qual fará a conexão aos outros servidores para acesso ao serviço de banco de dados. Estas conexões são todas gerenciadas através do mecanismo RMI.

O mecanismo RMI permite o acesso a um objeto em uma máquina diferente o qual é chamado de objeto remoto. Quando o cliente quer executar um método remoto ele simplesmente invoca um método normal da linguagem Java que será encapsulado em um objeto substituto chamado *stub*.

O *stub* empacota, como um bloco de bytes, os parâmetros utilizados no método remoto. O *stub*, então, envia estas informações para o servidor. No lado do servidor, um objeto receptor separa os parâmetros, localiza o objeto chamado, invoca o método e, por fim, envia um pacote composto dos dados de retorno para o cliente. O *stub* cliente separa o valor de retorno [Horstmann, 2001].

A conexão local entre o banco de dados e o serviço RMI é feita através da API Java *DataBase Connectivity* (JDBC). Utilizando JDBC é possível fazer acesso a um banco de dados independentemente do verdadeiro mecanismo de banco de dados que está sendo utilizado para armazenamento das informações [Reese, 2000], promovendo a heterogeneidade.

O conjunto de classes que implementam as interfaces JDBC para um determinado mecanismo de banco de dados é chamado *driver* JDBC. A figura 2, demonstra o modelo caracterizando, justamente, a existência de uma estrutura RMI para cada nó que disponibiliza o serviço de acesso ao banco de dados. Cada serviço conecta-se, localmente, a cada cópia do banco via um *driver* JDBC.

O cliente, por sua vez, executa uma *applet* a qual dispõe de um serviço RMI para acesso a objetos remotos. Conseqüentemente, a *applet* via RMI faz conexão com o serviço gerenciador local que se conecta aos os nós para consulta e/ou atualização das informações. A utilização do serviço local intermediário ocorre devido à limitação de conexão das *applets* com outros servidores que não o de onde fora carregada a *applet*. Portanto, é executada uma solicitação inicial de conexão RMI *applet* → serviço local bem com n (dependendo do número de nodos) conexões RMI serviço local → serviços

local e remotos de banco de dados.

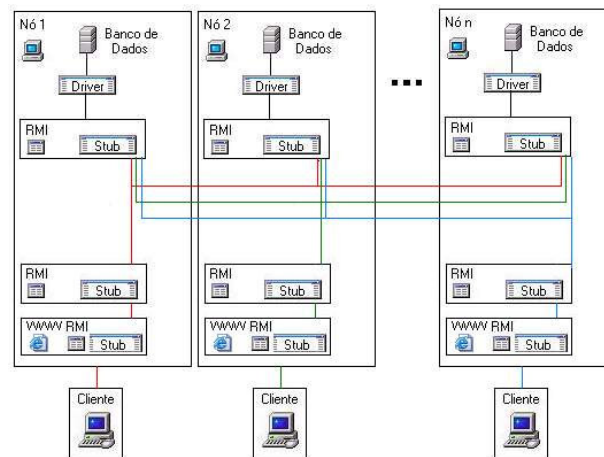


Fig. 2. Modelo Interno do Sistema de Banco de Dados Distribuído Replicado

Com relação ao ambiente estão sendo efetuados testes sobre dois bancos de dados: MySQL e PostgreSQL, utilizando plataformas Windows e Linux. O objetivo é que cada nó possua uma configuração diferenciada, ou seja, o nó 1, por exemplo, contenha plataforma Windows com MySQL, o nó 2 plataforma Windows com PostgreSQL, o nó 3 plataforma Linux com MySQL e assim sucessivamente. Tem-se por meta avaliar, futuramente, a performance, velocidade e segurança destes nós, em se tratando de que estes apresentam características bem diferenciadas.

3. Arquitetura

A arquitetura de uma aplicação focaliza o processamento do sistema de particionamento, ou seja, a arquitetura decide em que máquina e em que espaço de processo um determinado código deve executar [Özsu, 1999]. Qualquer aplicação de banco de dados é uma aplicação cliente/servidor se ela manipular armazenamento e recuperação de dados no processo de banco de dados e manipulação e apresentação de dados em qualquer outro lugar. O objetivo desta proposta é utilizar uma arquitetura de três camadas (arquitetura que utiliza um servidor de aplicação entre cliente e banco de dados) com vários servidores conectados a bancos de dados idênticos (replicados). Para que esta estrutura funcione corretamente a seguir serão demonstradas algumas características estabelecidas no lado do cliente e do servidor.

3.1. Cliente-Applet

A camada de apresentação, a qual caracteriza o cliente, deve fornecer uma interface de usuário para utilização do sistema. O cliente, na verdade, acessa uma máquina servidora, onde está instalado o serviço *Web*, por meio do software *Apache*. O serviço disponibiliza o acesso a *Applet* de gerenciamento. A *Applet* disponibiliza a interface onde são solicitadas as transações. Assim é feita a localização do serviço através da mensagem *lookup* identificando um serviço RMI de nome *AppletDBService*. Caso a conexão for bem sucedida é enviada a mensagem *retornaInformacao* passando como parâmetro o comando SQL requerido.

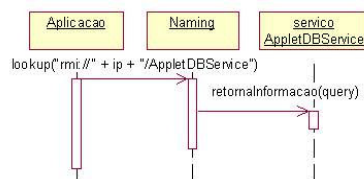


Fig. 3. Conexão Applet – Serviço Local

3.2. Serviço Local

Conforme as informações solicitadas pela *Applet* responsável pela aplicação, o sistema irá ativar várias linhas de execução (*threads*), no objetivo de consultar ou atualizar os diversos servidores. Através do envio da mensagem *start* a aplicação ativa cada *thread* para uma lista de servidores disponíveis em arquivo.

ipRemoto1=200.132.78.158
 ipRemoto2=200.132.78.146
 ipRemoto3=200.132.78.147
 ipRemoto4=end

soRemoto1=windows
 soRemoto2=linux
 soRemoto3=windows
 soRemoto4=end

Cada linha é colocada em execução através do método *run* cuja primeira solicitação é a execução do comando PING. O comando PING tem por função verificar se o servidor solicitado encontra-se ativo ou não. Caso o servidor solicitado encontre-se ativo é feita a tentativa para conexão com o serviço remoto. O primeiro passo é a localização do serviço através da mensagem *lookup* identificando um serviço RMI de nome *DataBase*. Por fim, se a conexão for bem sucedida é enviada a mensagem *executaComando* passando como parâmetro o comando SQL requerido. Neste momento o controle operacional está de posse do servidor. É importante salientar que a ativação das *threads* é feita para todos os servidores disponíveis.

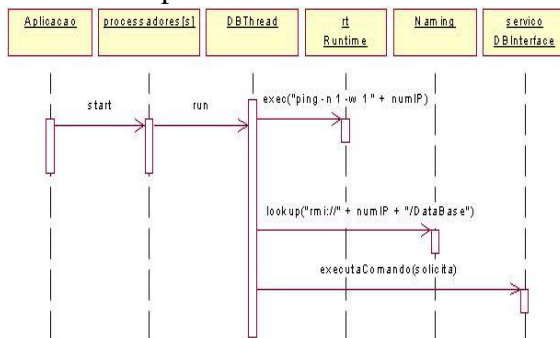


Fig. 4. Conexão Serviço Local – Serviço Distribuído

No caso de um comando SQL do tipo SELECT a primeira *thread* que atender a requisição do usuário tem por objetivo prover o cancelamento das outras que ainda estão tentando localizar a informação. No caso de comandos de atualização como INSERT, UPDATE ou DELETE cada *thread* deve terminar naturalmente.

Para os comandos de atualização são verificados quais os serviços que não estão disponíveis. Para cada serviço ativo será enviado o comando SQL a ser executado e o número IP da máquina que não oferece, temporariamente, o serviço. Estas informações são armazenadas no arquivo de *log* de cada servidor ativo.

A ativação das linhas de execução para envio das informações de falha ocorre de

maneira semelhante às consultas. A única diferença no diagrama da figura 3 é a modificação do método *executaComando(solicita)* pelo método *atualizaLog(numIPLog, solicita)*. Através do envio da mensagem *start* a aplicação, agora, ativa cada *thread* para uma lista de servidores que estão ativos no momento. Cada servidor recebe, portanto, a mensagem *atualizaLog* a qual é responsável pelo armazenamento destas informações no lado do servidor. Estas informações são armazenadas até o momento em que a máquina de número IP determinada voltar as suas atividades normais e permitir a atualização de sua base de dados.

3.3. Servidor

O lado servidor tem por responsabilidade gerenciar, fornecer e manipular as informações junto ao banco de dados. Os métodos remotos são definidos no lado do servidor sendo que o cliente só dispõe das interfaces dos mesmos. Portanto, os métodos *executaComando* e *atualizaLog* citados anteriormente são implementados no lado do servidor. Para cada solicitação de comando SQL recebida pelo servidor este precisa acessar a base local disponível. O acesso é feito via API JDBC que promove a conexão de uma aplicação Java a uma base de dados.

A figura 5, a seguir, mostra a seqüência do envio de mensagens para o método *executaComando*. Para cada acesso realizado ao serviço a aplicação faz uma nova conexão à base de dados. O método *getConnection* tem por objetivo atualizar as informações a respeito da base de dados que será utilizada. É feita a leitura de um arquivo texto o qual possui informações a respeito do *driver*, *url*, *username* e *password*. Esta facilidade foi inserida no intuito de propor ao usuário a utilização de qualquer tecnologia de base de dados no lado servidor desde que este possua os *drivers* de gerenciamento para tal.

```
jdbc.driver=org.gjt.mm.mysql.Driver
jdbc.url=jdbc:mysql://localhost/bancodados exemplo
jdbc.username=PUBLIC
jdbc.password=
```

Após a identificação das informações, quando uma classe *Driver*, reconhece sua URL, ela cria uma conexão de banco de dados utilizando as propriedades especificadas. Ela fornece a classe *DriverManager* uma implementação *java.sql.Connection* representando esta conexão de banco de dados. *DriverManager* retorna este objeto *Connection* para a implementação.

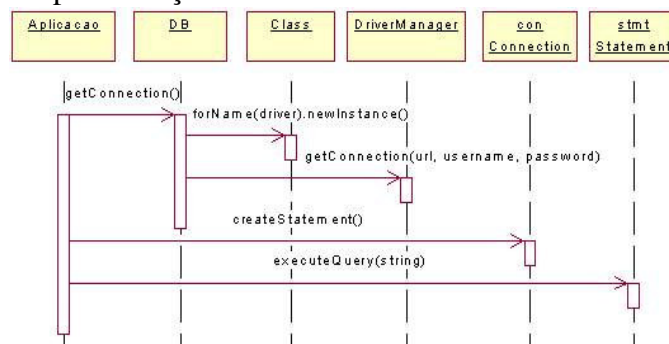


Fig. 5. Conexão Servidor

A partir do objeto *Connection* é criado um conjunto de procedimentos através de *createStatement* para que se possa fazer o direcionamento do comando SQL desejado. Por fim, o método *executeQuery* é responsável pela execução do comando desejado. É importante observar que para comandos do tipo *SELECT* o método retornará informações que serão encaminhadas ao lado cliente via serviço RMI. Para comandos de atualização que não geram informações de retorno para visualização, faz-se o retorno do status para gerenciamento das atividades.

O lado servidor comporta-se como uma aplicação com uma base de dados local. Caso a base de dados não esteja disponível no momento, pelo fato do banco de dados não estar ativo, o serviço RMI deste servidor mantém um arquivo de *log* com as operações de atualização. Toda vez que for proposta alguma verificação ou atualização de informações no banco e existirem informações no arquivo de *log* este é automaticamente atualizado.

Uma outra característica do lado servidor é que quando este ativado é feita uma procura nos *logs* disponíveis nos outros servidores se não existem atualizações a serem executadas para tanto é armazenado em arquivo as informações a respeito dos outros servidores.

dbc.ip=200.132.78.158

jdbc.so=windows

jdbc.ipRemoto1=200.132.78.146

jdbc.ipRemoto2=200.132.78.147

jdbc.ipRemoto3=end

jdbc.soRemoto1=linux

jdbc.soRemoto2=windows

jdbc.soRemoto3=end

3.4. Falha de Serviço

O grande problema a ser administrado pelo sistema é quando um determinado serviço não está disponível e comandos SQL de atualização são solicitados. Conforme mencionado na seção 3.2 o serviço local responsabiliza-se em detectar quais serviços não estão disponíveis enviando para os servidores ativos informações sobre os IPs com problemas bem como informando, também, os respectivos comandos SQL. Esta parte já está implementada e atende as necessidades previstas para o perfeito funcionamento do sistema.

Também implementada está a atualização das informações quando um determinado serviço retorna ao perfeito funcionamento. É importante salientar que um servidor não pode prover informações até que venha a ser totalmente atualizado. A idéia é que o serviço ao retornar a funcionar envie solicitações aos outros servidores verificando se estes outros detêm comandos em seus arquivos de logs. O pior caso é quando todos os outros servidores não fornecem o serviço. Neste caso o sistema deve interromper as suas atividades de atualização, fornecendo, apenas, informações sobre consultas, apresentando ao usuário que está atualizado até uma determinada data. Caso contrário cada servidor disponível retornará as informações que detém para atualizar a base em questão.

Uma outra questão que pode acontecer é a existência de duplicidades de informações nos arquivos de logs disponíveis. O gerenciamento desta etapa propõe o armazenamento no lado do servidor dos comandos já atualizados, com respectivas datas e horários de solicitação. Caso um servidor forneça as informações e outro esteja fora de

serviço, e também contenha as informações, em uma próxima atualização estes comandos já estarão atualizados e serão descartados.

Conclusões

Os mecanismos de suporte ao serviço RMI permitem a conexão entre diversas aplicações Java em equipamentos diferentes exceto para aplicações *Applets* executadas dentro de um navegador. Levando em consideração tal restrição implementou-se um serviço local que propicia a comunicação com outros equipamentos. As implementações propostas até o momento demonstram a eficácia e aplicabilidade da distribuição dos serviços remotos. Unindo, portanto, os serviços remotos aos serviços disponíveis para bancos de dados foram demonstrados, neste trabalho, o gerenciamento distribuído das atividades de consulta e atualização a bases de dados.

Preliminarmente, percebe-se que o estado atual do sistema atende aos requisitos no que se refere a consultas e atualizações. Com relação às atualizações, em caso de falha nos servidores, a implementação mantém atualizadas todas as bases disponíveis. Acredita-se que a validação da consistência lógica apresentada pelo sistema proposto será aumentada gradualmente com novos testes, portanto mais estudos de caso, envolvendo aplicações reais, estão sendo preparados para certificar o modelo.

Trabalhos Futuros

O objetivo deste trabalho foi demonstrar as características básicas de uma proposta para implementação de um sistema de banco de dados distribuído e replicado utilizando serviço RMI-Java, atualizado através de *Applets* Java. Alguns problemas ainda devem ser estudados. Com relação ao crescimento da rede são raros os estudos de desempenho de um SGBD distribuído. Faz-se importante detalhar a rede de comunicação observando o comportamento de protocolos de comunicação e de algoritmos à medida que os sistemas crescem.

Referências

- Breitbar, Y; Silberschatz, A. "Overview of Multidatabase Transaction Management", The VLDB Journal, vol. 1, n. 2, pp. 181-239, 1992.
- Chung, C; Dataplex, C. "An Accesss to Heterogeneous Distributed Databases", Communications of the ACM, vol. 33, n. 1, pp. 70-80, Jan. 1990.
- Elmasri, R; Navathe, S. Fundamentals of Database Systems, Benjamin/Cummings, 2a. edição, 1994.
- Horstmann, C; Cornell G. Core Java 2: Volume II, Advanced Features. [S.l.]: Prentice Hall, 2001.
- Korth, H; Silberschatz, A, Sistema de Banco de Dados. Makron Books, São Paulo, 1995.
- Lima, J.C; Ribeiro C. Oliveira, "Acesso Integrado a Banco de Dados Distribuídos Heterogêneos utilizando CORBA" in Proc. 1997 XII Simpósio Brasileiro de Banco de Dados, Fortaleza, Brasil, pp.333-348.
- Niemeyer, P; Peck, J. Exploring Java, [S.l.]: O'Reilly & Associates, 1997.
- Özsu, M.T; Valduriez, P. Principles of Distributed Database Systems, Prentice Hall, 2a. edição, 1999.
- Ram, S. "Heterogeneous Distributed Database Systems". Computer, v.24 , n.1, p.7-9, Dec. 1991.
- Reese, G. Programação para Banco de Dados. JDBC e Java, Ed. Bercley, São Paulo, 2000.